# La programmation client-serveur

Olivier Aubert



## Sources

- `http://www.info.uqam.ca/~obaid/INF4481/A01/Plan.htm`
- `http://bat710.univ-lyon1.fr/~exco/COURS/clientserveur.html`
- `man 2 socket`
- `http://www.developerweb.net/sock-faq/` FAQ socket
- RFC 791 (IP), RFC 793 (TCP), RFC 768 (UDP)
- Richard Stevens, *UNIX Network Programming*



## Le modèle client-serveur

- Communication entre processus : un client, l'autre serveur, qui coopèrent.
- Cette coopération s'effectue sous la forme d'un échange de données. Le client reçoit les résultats finaux délivrés par le serveur.



## Utilisation du modèle C/S

- S'étend de plus en plus vers tous les domaines d'activités :
  - World Wide Web
  - Gestion de bases de données
  - Systèmes transactionnels
  - Systèmes de messagerie
  - Systèmes de partage de données
  - Calcul scientifique
  - etc.

## *Les modes de connexion*

- Le mode *non connecté*
  - ne garantit pas :
    - l'intégrité des données
    - l'ordonnancement des données
    - la non-duplication des données
  - Ces propriétés doivent être garanties par l'application.
- Le mode *connecté* :
  - garantit les propriétés ci-dessus
  - implique une diminution des performances brutes par rapport au mode non-connecté. Peut constituer une contrainte.
  - Permet une implémentation asynchrone des échanges



## *Les clients*

- Une application cliente est moins complexe qu'une application serveur.
  - La plupart des applications clientes ne gèrent pas d'interactions avec plusieurs serveurs.
  - La plupart des applications clientes sont traitées comme un processus conventionnel ; un serveur peut nécessiter des accès privilégiés.



## *Les serveurs*

Le processus serveur :

- offre un point d'entrée sur le réseau
- entre dans une boucle infinie d'attente de requêtes
- à la réception d'une requête, déclenche les processus associés à la requête, puis émet la réponse vers le client
- Deux types de serveurs :
  - itératifs : ne gèrent qu'un seul client à la fois
  - parallèles : fonctionnent en mode concurrent



## *Choix du type de serveur*

- Serveurs itératifs en mode non-connecté :
  pour les services qui nécessitent très peu de traitements par requête. Par exemple, un serveur de temps.
- Serveurs itératifs en mode connnecté :
  pour les services qui nécessitent très peu de traitements par requête, mais ont besoin d'un transport fiable.
- Serveurs concurrents en mode non-connecté :
  - Temps de création d'un processus extrêmement faible par rapport au temps de traitement d'une requête.
  - Les requêtes nécessitent des accès périphériques importants.

## *Les sockets*

- Interface client/serveur utilisée à l'origine dans le monde UNIX et TCP/IP.
- Fournit les primitives pour le support des communications reposant sur le protocole IP.
- Les applications client et serveur ne voient les couches de communication qu'à travers l'API *socket*.



## *Définition des sockets*

- Les sockets peuvent être vues comme un point d'accès par lequel un processus peut envoyer ou recevoir des données.
- La socket se présente sous la forme d'un descripteur de fichier.
- Les sockets sont une interface d'accès au réseau au dessus du service transport. C'est une interface souple mais de bas niveau.



## *Structure des sockets*

- Il existe différents types de sockets déterminées selon les propriétés de la communication que l'on veut établir.
- Les propriétés que l'on peut choisir sont :
  - respect de la séquentialité des envois
  - éviter les doubles envois
  - messages à caractère urgent
  - mode connecté



## *Caractéristiques des sockets*

Une socket possède trois caractéristiques :

- **Type** Décrit la manière dont les données sont transmises : `SOCK_DGRAM`, `SOCK_STREAM`, `SOCK_RAW`.
- **Domaine** Définit la nommage des sockets et les formats d'adressage utilisés : `AF_UNIX`, `AF_INET`, `AF_INET6`, `AF_X25`.
- **Protocole** Décrit le protocole de communication à utiliser pour émettre et recevoir les données. Généralement déterminé par le domaine : `PF_UNIX`, `PF_INET`, `PF_INET6`, `PF_X25`.

## Les domaines des sockets

- Le domaine d'une socket permet de spécifier le mode d'adressage et l'ensemble des protocoles utilisables par la socket.
- Les domaines d'adressage les plus courants sont Unix (`AF_UNIX`) et IP (`AF_INET`), voire IPv6 (`AF_INET6`).



## Le domaine d'adressage

- Domaine `AF_UNIX` : pour les processus qui communiquent sur la même machine.
- Domaine `AF_INET` : pour les processus qui communiquent à travers le réseau IP. Les protocoles disponibles sont TCP/IP ou UDP/IP, voire IP.



## Adressage

- Lorsqu'un processus serveur veut fournir des services, il a besoin d'un port d'attache pour la localisation de ses services.
- Un certain nombre de numéros est réservé aux services standards.
- Pour le serveur, les ports inférieurs à 1024 ne peuvent être utilisés que par le super-utilisateur `root`.
- Les numéros de port standard sont définis dans le fichier `/etc/services`, et accessibles via la commande `getservbyname()`.



## Structures de socket

Les structures `sockaddr` et `sockaddr_in` sont définies comme suit :

```
struct sockaddr {
        sa_family_t      sa_family;       /* domaine */
        char             sa_data[14];     /* adresse ef
}

struct sockaddr_in {
  sa_family_t            sin_family;      /* domaine
  unsigned short int     sin_port;        /* numro de po
  struct in_addr         sin_addr;        /* adresse IP
  unsigned char          __pad[__SOCK_SIZE__ - ...] /*
}
```

## Création d'une socket

On crée une socket avec la primitive `socket()` :

```
s = socket (AF_INET, SOCK_STREAM, 0);
```

`s` est l'identificateur de la socket. On peut l'utiliser pour les opérations de lecture/écriture.



## Gestion des connexions

- `bind()` permet d'associer une adresse à une socket :
  `int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)`
- `connect()` est utilisée par le client pour établir la connexion :
  `int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);`
- `listen()` est utilisée par le serveur pour établir la connexion :
  `int listen(int s, int backlog);`
- `accept()` est utilisée par le serveur pour indiquer qu'il est prêt à accepter des connexions. Appel généralement bloquant :
  `int accept(int s, struct sockaddr *addr, socklen_t *addrlen);`
- `shutdown()` et `close()` permettent de fermer une connexion :
  `int shutdown(int s, int how);`



## Transfert en mode connecté

- Les commandes `send()` et `recv()` permettent d'échanger des données :
  ```
  int send(int s, const void *msg, size_t len, int fl
  int recv(int s, void *buf, size_t len, int flags);
  ```
- On peut préciser des options dans les champs `flags`. Par exemple, `MSG_OOB` pour les données urgentes.
- Attention : `send()` peut n'envoyer qu'une partie des données. Ne pas oublier de vérifier sa valeur de retour...
- Il est également possible d'utiliser les appels `read()` et `write()` qui offre moins de possibilités de contrôle.



## Client-serveur en mode connecté

Dans ce mode, on établit d'abord une connexion. On peut ensuite transférer des données à travers le tuyau créé.